



**Tecnológico
de Monterrey**

REPORTE DE SOLUCIÓN DEL RETO

Hotel-ID to combat human trafficking

Armando de Jesús Cerdá de la Rosa A01570376

Carlos David Toapanta Noroña A01657439

Omar Enrique González Uresti A00827095

Josue Salvador Cano Martínez A00829022

Grecia Pacheco Castellanos A01366730

10 de noviembre del 2022

Inteligencia artificial avanzada para la ciencia de datos

Jorge Cruz | Daniel Otero | Blanca Ruiz | Félix Botello | César Guerra

ÍNDICE

ÍNDICE	2
VÍDEO	3
INTRODUCCIÓN	4
MARCO TEÓRICO	5
Convolutional Neural Network (CNN)	5
- ResNet	5
- DenseNet	5
- Efficient Net	5
KERAS	5
FASTAI	6
PYTORCH	6
PROPIUESTA DE SOLUCIÓN	7
SPLITTER	7
BALANCING	7
AUGMENTATION	8
Transformaciones geométricas	8
Transformaciones de espacio de color	10
MODEL TESTING	11
ResNet	11
DenseNet	11
Efficient Net	16
VGG16	17
OPTIMIZATION	18
Adam (ResNet18, Dataset: g30, 30 epoch)	18
QHAdam (ResNet18, Dataset: g30, 30 epoch)	20
AdamW (ResNet18, Dataset: g30, 30 epoch)	22
CONCLUSIONES	25
REFERENCIAS	26

VÍDEO

[Vídeo explicativo](#)



INTRODUCCIÓN

La trata de personas es un delito que se presenta en todas las regiones del mundo, donde día con día se priva de su dignidad a millones de personas. Combatir este delito demanda un esfuerzo colaborativo de múltiples organismos, tanto gubernamentales como no gubernamentales, nacionales e internacionales, que abarquen un amplio espectro de actividades de justicia penal, compromiso judicial, derechos humanos y desarrollo.

En muchas de las ocasiones las víctimas de trata de personas son fotografiadas en habitaciones de hoteles, por lo cual, identificar los lugares donde fueron tomadas las fotos es de vital importancia; lamentablemente esto representa una gran cantidad de retos debido a la baja calidad de las imágenes y los ángulos poco comunes de las mismas.

Por lo anteriormente expuesto, el presente trabajo busca poder generar un modelo capaz de identificar los hoteles en los que fueron tomadas las imágenes pertenecientes al set de datos “TraffickCam”, el cual está basado en una galería de imágenes de entrenamiento con los identificadores de los hoteles conocidos.

MARCO TEÓRICO

Convolutional Neural Network (CNN)

Una red neuronal convolucional (CNN o ConvNet) es una clase de redes neuronales profundas. Las CNN se emplean más comúnmente en la visión por computadora. Dada una serie de imágenes o videos del mundo real, con la utilización de CNN, el sistema de IA aprende a extraer automáticamente las características de estas entradas para completar una tarea específica, por ejemplo, clasificación de imágenes, autenticación de rostros y segmentación semántica de imágenes.

Tipos:

- ResNet

Aprende funciones residuales con referencia a las entradas de la capa, en lugar de aprender funciones no referenciadas. En lugar de esperar que cada una de las capas apiladas se ajuste directamente a un mapeo subyacente deseado, las redes residuales permiten que estas capas se ajusten a un mapeo residual.

- DenseNet

Utiliza conexiones densas entre capas, a través de bloques densos, donde conectamos todas las capas (con tamaños de mapa de características coincidentes) directamente entre sí. Para preservar la naturaleza de avance, cada capa obtiene entradas adicionales de todas las capas anteriores y pasa sus propios mapas de características a todas las capas posteriores.

- EfficientNet

Es una arquitectura de red neuronal convolucional y un método de escalado que escala uniformemente todas las dimensiones de profundidad/ancho/resolución utilizando un coeficiente compuesto. A diferencia de la práctica convencional que escala arbitrariamente estos factores, el método de escala EfficientNet escala uniformemente el ancho, la profundidad y la resolución de la red con un conjunto de coeficientes de escala fijos.

KERAS

Keras es una API de aprendizaje profundo escrita en Python, que se ejecuta sobre la plataforma de aprendizaje automático TensorFlow. Fue desarrollado con un enfoque en permitir la experimentación rápida. Ser capaz de pasar de la idea al resultado lo más rápido posible es clave para hacer una buena investigación.

FASTAI

Fastai es una biblioteca de aprendizaje profundo que ofrece componentes de alto nivel que pueden proporcionar rápida y fácilmente resultados de última generación en dominios de aprendizaje profundo estándar, y proporciona componentes de bajo nivel que se pueden mezclar y combinar para crear nuevos enfoques. Su objetivo es hacer ambas cosas sin comprometer sustancialmente la facilidad de uso, la flexibilidad o el rendimiento.

PYTORCH

Es un paquete de Python diseñado para realizar cálculos numéricos haciendo uso de la programación de tensores. Además permite su ejecución en GPU para acelerar los cálculos.

PROPUESTA DE SOLUCIÓN

SPLITTER

El primer aspecto para lograr probar el entrenamiento de los diferentes modelos es realizar una separación del dataset de entrenamiento, debido a que Kaggle no nos proporciona un dataset para validar nuestro entrenamiento.

Para lograr lo anterior, primeramente se realizó un análisis del dataset donde se buscó obtener datos tal y como que tan balanceadas se encontraban las clases para lograr crear posteriormente una estrategia para balancearlas. A partir de este análisis inicial, se determinó que necesitábamos descartar clases con muy pocas imágenes, en este caso terminamos teniendo como 15 el mínimo número de imágenes que una clase necesitaba para no ser descartada, esto con la finalidad de lograr un mejor entrenamiento del modelo.

Por último, se procedió a realizar el split del dataset en dos datasets, uno de train y uno de validation. Para esto nuevamente se necesitó determinar un porcentaje de split para separar las imágenes del dataset original en proporciones que ayuden a nuestro modelo entrenar de la mejor manera posible. Para esto llegamos al porcentaje de split de 90% para el dataset de entrenamiento (train), dejando 10% de las imágenes para el split de validación(validation). Ya teniendo estos dos datasets separados nos es posible realizar el balanceo de las clases en el dataset de train para mejorar el entrenamiento.

BALANCING

Con la finalidad de mejorar el rendimiento de los algoritmos de aprendizaje automático implementados, se hizo uso de balancing, donde el propósito consiste en obtener un conjunto de datos equilibrado (dataset donde cada clase de salida está representada por el mismo número de muestras de entrada).

Para el caso del reto, el propósito de entrenar un modelo para detectar los hoteles donde las víctimas han sido fotografiadas representa un escenario donde después del entrenamiento se obtiene alrededor de 95% de precisión, sin embargo, al ponerlo a prueba en sucesivas predicciones es posible identificar que el modelo falla. Esto es claramente un problema debido a que muchos algoritmos de aprendizaje automático están diseñados para maximizar la precisión general. Es aquí donde entra la necesidad de equilibrar los datos, donde no existe una diferencia muy alta entre los valores de las distintas clases de imágenes.

Imbalanced Class Distribution

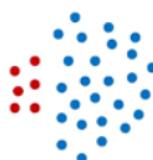


Imagen 1. Diagrama de representación de una clase no balanceada.

La métrica de evaluación que se utilizó para validar el grado de balance que tiene el set de datos es la precisión (resultante de dividir el número de verdaderos positivos entre todas las predicciones positivas) donde una baja precisión indica un alto número de falsos positivos. Como técnica de balanceo se optó por implementar sobrecolección, técnica donde se modificaron las clases de datos desiguales para crear conjuntos de datos equilibrados, es decir, se aumentó el número de miembros de las clases minoritarias en el conjuntos de entrenamientos tomando como estrategia procesos de data augmentation. La principal ventaja fue no haber perdido información (lo cual sucede, por ejemplo, con el submuestreo).

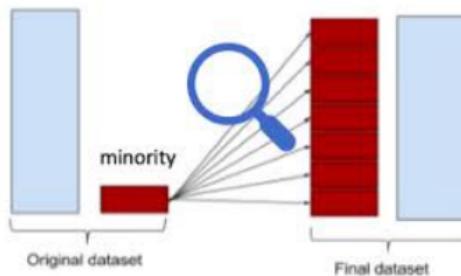


Imagen 2. Diagrama de representación de las clases balanceadas.

Lo anterior permitió obtener un set de datos balanceado entre las diferentes clases, obteniendo así una mayor precisión del modelo, reflejándose en una mejor predicción del mismo en los test ejecutados.

AUGMENTATION

Con el objetivo de tener un set de datos más variado y por ende más completo para que los modelos tengan un mejor ajuste se aplicaron de manera aleatoria diferentes tipos de transformaciones a las imágenes que ya se tenían en el set de datos. El tipo de transformaciones que se realizaron fueron geométricas y de transformaciones del espacio de color.

Este proceso se enfocó principalmente en la realización de transformaciones geométricas debido a que estas son las más adecuadas cuando se tiene un sesgo posicional en las imágenes ; las modificaciones realizadas fueron basadas en rotación, traslación, recorte, reflexión, ajuste de cizallamiento y escalamiento.

Por otro lado, debido a que puede haber variaciones de iluminación , se realizaron transformaciones de cambio de canal de color y ajuste de brillo.

Transformaciones geométricas

1. Rotación: En esta transformación se especifica el ángulo de rotación en grados en un rango de 0 a 360, donde se especifica este mismo como un número entero.Un ejemplo del efecto de esta transformación puede notarse en la imagen 3.



Imagen 3. Ejemplo de las transformaciones por rotación.

2. Escalamientos: En este caso se realizaron dos modificaciones, cambio de altura y cambio de anchura; debido a que en ocasiones los objetos no se van a encontrar en el centro de la imagen, por lo cual se hacen estas modificaciones verticales y horizontales en los píxeles de la misma. En ambos casos se especifica como un flotante entre 0 al 1 el porcentaje de altura y anchura que se tiene que desplazar. Podemos observar ejemplos de estas modificaciones en las imágenes 4 y 5 .



Imagen 4. Ejemplo de las transformación por cambio de anchura.



Imagen 5. Ejemplo de las transformación por cambio de altura.

3. Reflexiones: Estas reflexiones se realizan sobre ambos ejes, donde se revierten las filas y columnas de píxeles respectivamente, para especificar que se requiere esta transformación se envía el valor booleano para cada dirección de reflexión. A continuación, en la imagen 6, se muestra el resultado de esta transformación.



Imagen 6. Ejemplo de las transformación por reflexión.

4. Zoom: Transformación relacionada con el acercamiento y alejamiento de la imagen, dependiendo del valor que se le asigne; donde los valores menores a cero realizan el acercamiento, y los valores mayores a este el alejamiento.

Para el caso que se está manejando e identificar de mejor manera aspectos de las habitaciones es más recomendable el acercamiento por lo cual el rango de zoom usado es negativo.



Imagen 7]. Ejemplo de las transformación por zoom.

5. El ajuste de cizallamiento: Considerando que el cizallamiento es el desplazamiento de cada punto en una dirección fija en una proporción a una distancia; el parámetro especifica la cantidad de ángulos que la imagen será modificada en sentido dextrógiro. Este tipo de modificaciones es útil en caso de que los componentes de la imagen no estén orientados en la dirección adecuada.

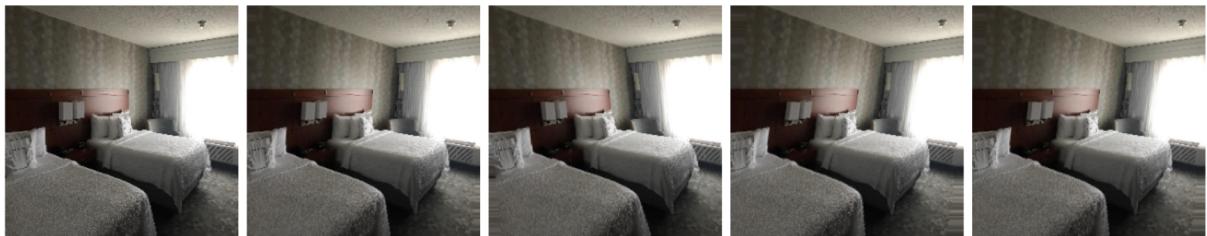


Imagen 8]. Ejemplo de las transformación por ajuste de cizallamiento.

Transformaciones de espacio de color

Tomando en cuenta de que consideraciones lumínicas y de concentración de color podrían variar en la toma de las fotos, se realizaron dos transformaciones.

1. Ajuste de brillo: Es una transformación base debido a que la mayoría del tiempo no se tienen las mismas condiciones de luminosidad, por lo cual es necesario ajustar la cantidad el rango de la cantidad de brillo de las imágenes. En este caso el rango se encuentra entre valores del 0 al 1; donde el 0 indica que no tiene brillo y el 1 es el nivel máximo del mismo. Al observar la imagen 9, se notan las variaciones de este aspecto sobre una misma imagen ejemplo.



Imagen 9. Ejemplo de las transformación por ajuste de brillo.

2. Cambio de canal: Esta transformación consiste en la separación de los canales de color y el intercambio de estos de acuerdo con un número aleatorio dentro de un rango aleatorio.



Imagen 10. Ejemplo de las transformación por cambio de canal..

MODEL TESTING

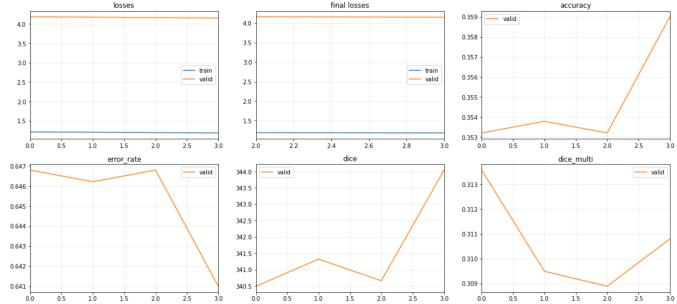
Con la finalidad de asegurarnos de obtener los mejores resultados realizamos pruebas sobre diferentes modelos pre entrenados para encontrar la arquitectura que mejor desempeño demuestre basándonos en diferentes métricas tal y como la precisión del modelo, tasa de error, pérdidas, etc...

Presentaremos los resultados de los diferentes modelos probados a continuación:

ResNet-50

	precision	recall	f1-score	support
0	0.00	0.00	0.00	2
1	0.14	0.50	0.22	2
2	1.00	0.50	0.67	2
3	0.00	0.00	0.00	2
4	0.00	0.00	0.00	2
5	1.00	0.00	0.00	2
6	1.00	0.20	0.33	5
7	0.25	0.50	0.33	2
8	0.50	0.75	0.60	4
9	1.00	0.50	0.67	2
10	0.50	0.33	0.40	3
11	0.00	0.00	0.00	2
12	0.00	0.00	0.00	2
13	0.50	0.57	0.53	7
14	1.00	1.00	1.00	2
15	0.00	0.00	0.00	3
16	1.00	0.14	0.25	7
17	0.00	0.00	0.00	2
18	1.00	0.00	0.00	3
19	1.00	0.00	0.00	2
20	0.33	0.50	0.40	2
21	0.00	0.00	0.00	5
22	0.25	0.67	0.36	3
...				
accuracy			0.36	1727
macro avg	0.56	0.36	0.31	1727
weighted avg	0.57	0.36	0.33	1727

Es un tipo de red neuronal convolucional presentada en 2015, que tiene 50 capas. 48 convolucionales, 1 de Maxpool y 1 de Average Pool. Ahora, del lado del entrenamiento se ocuparon 3 épocas de fine tune, 4 de entrenar todas las capas con nuevo learning rate, 12 de todas las copas con otro learning rate para poder llegar una accuracy de 36% en el dataset.



DenseNet

DenseNet es un tipo de red neuronal convolucional que utiliza conexiones densas a través de bloques densos los cuales conectan todas las capas directamente entre ellas. Es así como cada capa recibe conocimiento colectivo de todas las capas pasadas y esto se puede observar a sí mismo en el proceso de backpropagation. Es por esto que esta arquitectura suena muy atractiva para resolver este reto.

Entrenamiento base:

El entrenamiento base consiste en importar el modelo pre-entrenado y posteriormente entrenar el modelo utilizando los datos de entrenamiento sin balancear y sin algún tipo de ajuste de hiper parámetro. En este caso se realizó el entrenamiento durante 8 épocas y se llegó a un accuracy de 0.537, el cual es bastante bueno pero todavía se puede mejorar.

epoch	train_loss	valid_loss	accuracy	error_rate	dice	time
0	5.475326	4.791102	0.268450	0.731550	453.304973	06:00
epoch	train_loss	valid_loss	accuracy	error_rate	dice	time
0	4.258882	4.254540	0.320980	0.679020	461.685580	07:32
1	4.086619	3.889636	0.363519	0.636481	472.194922	07:33
2	3.429395	3.498754	0.406381	0.593619	467.284307	07:34
3	2.785995	3.132929	0.459233	0.540767	482.651071	07:34
4	1.955179	2.901911	0.494683	0.505317	480.914307	07:34
5	1.303827	2.735166	0.524009	0.475991	489.687749	07:32
6	0.873526	2.656617	0.540445	0.459555	490.797837	07:33
7	0.680219	2.657590	0.537867	0.462133	490.639417	07:31

Entrenamiento optimizado

Con la finalidad de mejorar el entrenamiento se buscaron diferentes maneras de optimizarlo, para esto se llevó a cabo una investigación de diferentes funciones de pérdida y optimizadores, en este caso se entrenó el mismo modelo pero en este caso se utilizó la función de pérdida LabelSmoothingCrossEntropy y la función de optimización QHAdam.

Con estos ajustes y un aumento del número de épocas de 8 a 14 se comenzó el entrenamiento, en nuestra época 8 ya se logró notar una mejora de 1% y para nuestra época 14 terminamos con un accuracy de 59.6%. La justificación de la mejoría de estos resultados fue que al utilizar la función de pérdida comenzamos con un accuracy un poco más bajo pero conforme fueron pasando las épocas fuimos obteniendo un aprendizaje mejor sin que parara de aprender, a su vez en esto ayudó la función de optimización QHAdam al mejorar el ajuste de los pesos, en conjunto logramos mejorar el entrenamiento y obtener un accuracy 5% más alto.

epoch	train_loss	valid_loss	accuracy	error_rate	dice	dice_multi	time
0	6.030476	5.259388	0.264260	0.735740	467.948575	0.032256	06:56
epoch	train_loss	valid_loss	accuracy	error_rate	dice	dice_multi	time
0	4.985419	4.823584	0.301644	0.698356	469.228295	0.050903	06:26
1	4.523584	4.503068	0.346117	0.653883	469.302329	0.093066	06:48
2	4.056201	4.210901	0.388656	0.611344	477.076819	0.138115	06:27
3	3.699290	3.939758	0.427006	0.572994	477.253405	0.186162	06:28
4	3.158849	3.694500	0.475346	0.524654	475.433682	0.240696	06:18
5	2.874310	3.556392	0.504995	0.495005	487.134335	0.286150	06:18
6	2.568599	3.458421	0.522720	0.477280	483.982335	0.317872	06:14
7	2.191221	3.359851	0.545923	0.454077	489.251792	0.345898	06:21
8	1.966095	3.327547	0.552691	0.447309	489.608363	0.355907	06:30
9	1.796441	3.252904	0.560748	0.439252	491.924202	0.356316	06:30
10	1.667392	3.205577	0.576539	0.423461	493.210763	0.377769	06:21
11	1.592688	3.161315	0.588785	0.411215	493.985100	0.397080	06:27
12	1.555047	3.151394	0.591363	0.408637	494.451391	0.398390	06:33
13	1.501972	3.129047	0.592330	0.407670	495.903248	0.399343	06:30
14	1.514680	3.136121	0.596197	0.403803	495.685726	0.404865	06:57

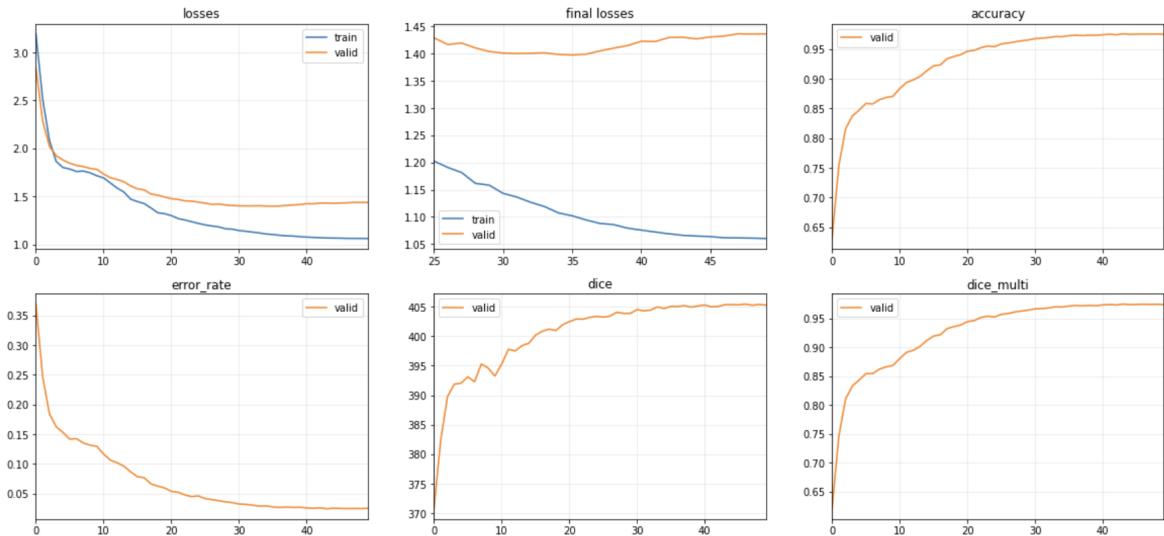
Entrenamiento optimizado y con el augmented dataset

Por último, el cambio que tuvo mayor efecto en el entrenamiento fue sin duda alguna el balanceo de imágenes. Al balancear el número de imágenes por clase utilizando en conjunto técnicas de data augmentation se logró enriquecer el entrenamiento con un impacto increíble, de esta forma es que logramos obtener un accuracy inicial en nuestra primer epoca de entrenamiento de 63% debido a que se expuso a una cantidad mayor de imágenes modificadas para resaltar características importantes de cada imagen original.

epoch	train_loss	valid_loss	accuracy	error_rate	dice	dice_multi	time
0	3.192012	2.839636	0.630861	0.369139	370.738521	0.620192	09:44
1	2.512856	2.285602	0.753429	0.246571	382.420182	0.745534	09:41
2	2.081963	2.020386	0.815789	0.184211	389.783492	0.810366	09:42
3	1.862193	1.922993	0.837640	0.162360	391.873464	0.833099	09:41
4	1.799725	1.877709	0.847209	0.152791	392.039193	0.843212	09:42
5	1.784073	1.844347	0.858533	0.141467	393.114799	0.854418	09:45
6	1.758251	1.821653	0.857735	0.142265	392.289266	0.854551	09:44
7	1.763262	1.809579	0.864912	0.135088	395.298756	0.862283	09:45
8	1.744725	1.790944	0.868660	0.131340	394.605867	0.866243	09:44
9	1.714328	1.780691	0.870574	0.129426	393.247966	0.868725	09:46
10	1.691918	1.732740	0.883652	0.116348	395.287100	0.880487	09:46
11	1.638917	1.693218	0.893939	0.106061	397.792286	0.891451	09:46
12	1.586171	1.675833	0.898325	0.101675	397.509393	0.895024	09:45
13	1.544324	1.650863	0.904147	0.095853	398.407206	0.901765	09:45
14	1.470420	1.607709	0.913716	0.086284	398.801389	0.911950	09:45
15	1.445306	1.578370	0.921930	0.078070	400.179697	0.919668	09:47
16	1.424539	1.566327	0.923604	0.076396	400.854130	0.921886	09:45
17	1.378682	1.524370	0.934051	0.065949	401.206366	0.932549	09:46
18	1.329036	1.512114	0.937719	0.062281	400.971638	0.936020	09:45
19	1.319272	1.492739	0.940750	0.059250	401.944869	0.938787	09:46
20	1.299520	1.475552	0.946571	0.053429	402.471678	0.944671	09:47
21	1.268500	1.467394	0.948086	0.051914	402.913820	0.946494	09:48
22	1.253592	1.453024	0.952552	0.047448	402.889489	0.951948	09:49
23	1.235865	1.450223	0.955423	0.044577	403.192711	0.954131	09:55

24	1.217725	1.439205	0.954306	0.045694	403.373627	0.952636	09:54
25	1.202572	1.429176	0.958852	0.041148	403.237318	0.957341	09:55
26	1.190855	1.416726	0.960447	0.039553	403.417838	0.958620	09:53
27	1.181579	1.419507	0.962360	0.037640	404.066155	0.961393	09:50
28	1.161650	1.410469	0.964195	0.035805	403.875179	0.962896	09:49
29	1.158311	1.403944	0.965630	0.034370	403.864432	0.964788	09:50
30	1.143270	1.401085	0.967783	0.032217	404.526973	0.966748	09:50
31	1.136444	1.400350	0.968581	0.031419	404.308890	0.967333	09:51
32	1.126883	1.400630	0.969697	0.030303	404.431296	0.968420	09:51
33	1.119010	1.401588	0.971372	0.028628	404.970893	0.970234	09:43
34	1.107417	1.398517	0.971132	0.028868	404.708104	0.969983	09:43
35	1.102032	1.397529	0.972887	0.027113	405.068012	0.971806	09:40
36	1.094454	1.398906	0.973604	0.026396	405.047924	0.972503	09:42
37	1.088017	1.404969	0.972966	0.027033	405.198909	0.972114	09:41
38	1.085815	1.410105	0.973604	0.026396	404.937044	0.972696	09:42
39	1.079376	1.414723	0.973365	0.026635	405.141201	0.972334	09:43
40	1.075723	1.422842	0.974402	0.025598	405.295565	0.973585	09:44
41	1.072268	1.422233	0.975199	0.024801	404.978138	0.974249	09:43
42	1.068918	1.429637	0.974322	0.025678	405.061181	0.973492	09:41
43	1.066160	1.430097	0.975917	0.024083	405.358729	0.975133	09:43
44	1.064768	1.427193	0.974960	0.025040	405.348867	0.974077	09:49
45	1.063783	1.430676	0.975359	0.024641	405.318131	0.974505	09:49
46	1.061681	1.432382	0.975598	0.024402	405.457540	0.974841	09:49
47	1.061420	1.436424	0.975439	0.024561	405.246647	0.974539	09:51
48	1.060913	1.436053	0.975598	0.024402	405.381627	0.974767	09:51
49	1.060216	1.436309	0.975120	0.024880	405.283047	0.974271	09:52

Al final del entrenamiento, en nuestra época 50 se puede visualizar que obtuvimos como accuracy final 97%, una gran mejoría en este caso. A su vez contamos con gráficas en la imagen inferior donde podemos observar las diferentes métricas y podemos ver como por ejemplo nuestro loss fue decrementando y como nuestro accuracy fue mejorando conforme pasaban el número de épocas.



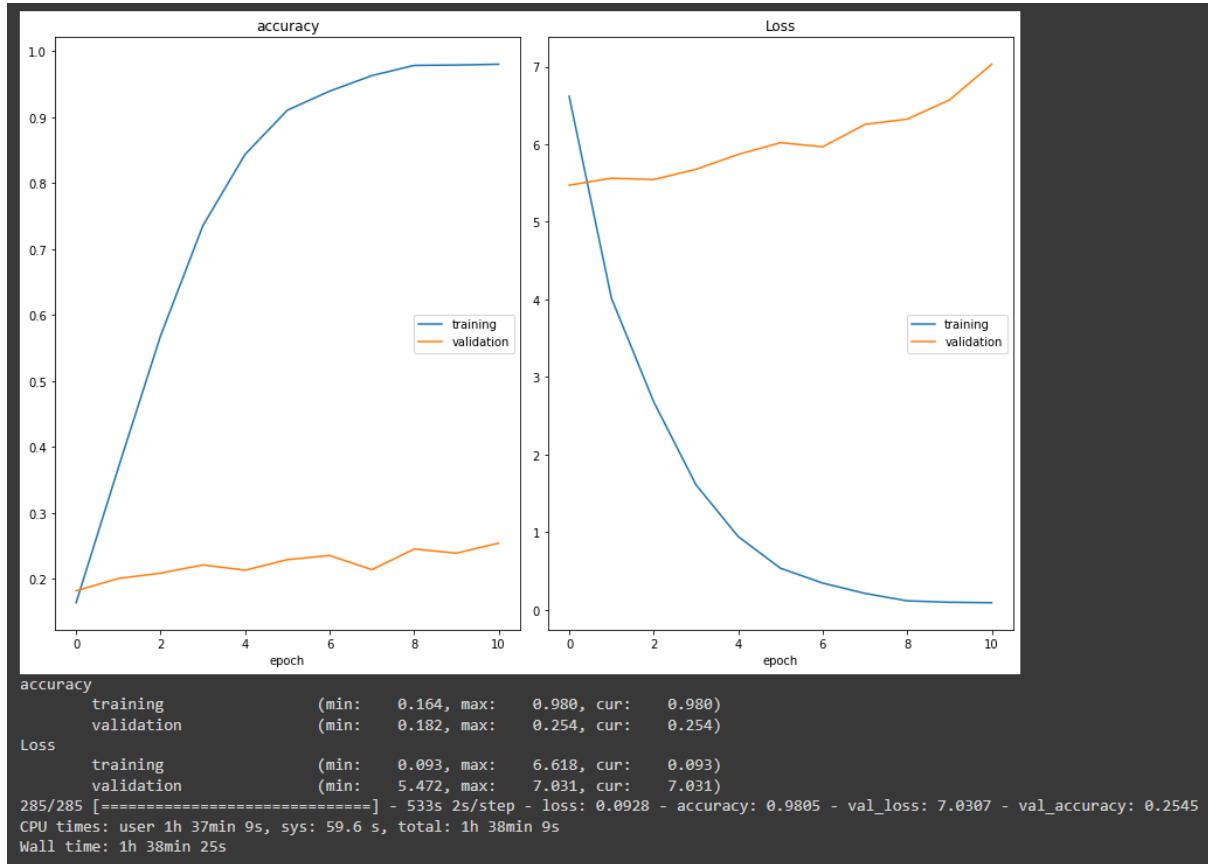
Al final se realizaron predicciones contra el validation dataset y se obtuvo de accuracy 59% macro average y 56% weighted average.

accuracy	
macro avg	0.59
weighted avg	0.56

VGG16

Se empleo una estrategia de Transfer Learning llamada Fine-Tuning. El objetivo de Fine-Tuning es permitir que una parte de las capas preentrenadas se vuelvan a entrenar. Se obtuvo el siguiente resultado:

Validation Accuracy: 0.2545



OPTIMIZATION

Se hizo una investigación de diferentes modelos de optimización basados en ResNet18. Utilizando diferentes datasets, así como diferente número de epoch para poder testear de diferente manera cada uno de los optimizadores que contemplamos para el proyecto. Cada uno de los optimizadores se probó con ResNet18 con un total de 30 épocas, para poder hacer una comparación más adecuada.

Adam (ResNet18, Dataset: g30, 30 epoch)

El optimizador base es un método para optimización estocástica que requiere gradientes de primer orden y baja memoria. Este tipo de optimización es adecuada cuando tenemos una gran cantidad de datos

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	3.347865	2.950589	0.443001	0.556999	06:23
1	2.891350	2.726490	0.471353	0.528647	06:18
2	2.533319	2.535841	0.509746	0.490254	06:26
3	2.198258	2.350671	0.533373	0.466627	06:32
4	1.803594	2.208826	0.556409	0.443591	06:27
5	1.463383	2.032915	0.581807	0.418193	06:29
6	1.151990	1.950112	0.600118	0.399882	06:24
7	0.897651	1.869547	0.622563	0.377436	06:29
8	0.662598	1.812009	0.636739	0.363260	06:36
9	0.482543	1.816494	0.642056	0.357944	06:28
10	0.364101	1.817334	0.647962	0.352038	06:31
11	0.265929	1.795929	0.650325	0.349675	06:25
12	0.214820	1.814004	0.653278	0.346722	06:21
13	0.180660	1.818019	0.659776	0.340224	06:16

13	0.180660	1.818019	0.659776	0.340224	06:16
14	0.149211	1.846479	0.658594	0.341406	06:26
15	0.113522	1.797816	0.666864	0.333136	06:22
16	0.089717	1.826509	0.665682	0.334318	06:25
17	0.071673	1.787718	0.676314	0.323686	06:32
18	0.068736	1.814752	0.669226	0.330774	06:38
19	0.053862	1.830235	0.668045	0.331955	06:43
20	0.040487	1.816045	0.668045	0.331955	06:27
21	0.035275	1.810476	0.672770	0.327230	06:28
22	0.036390	1.783074	0.676314	0.323686	06:27
23	0.025239	1.777017	0.677496	0.322504	06:27
24	0.024100	1.752979	0.680449	0.319551	06:28
25	0.020963	1.740595	0.683402	0.316598	06:24
26	0.016819	1.755497	0.682812	0.317188	06:35
27	0.018853	1.739148	0.684584	0.315416	06:34
28	0.013564	1.747370	0.683993	0.316007	06:37
29	0.013847	1.739709	0.684584	0.315416	06:36

QHAdam (ResNet18, Dataset: g30, 30 epoch)

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	3.331895	2.950334	0.441819	0.558181	06:38
1	2.893999	2.728036	0.472534	0.527466	06:38
2	2.520187	2.529162	0.507383	0.492617	06:45
3	2.168307	2.337748	0.537507	0.462493	06:44
4	1.807748	2.178767	0.559362	0.440638	06:45
5	1.459688	2.013179	0.593030	0.406970	06:46
6	1.144866	1.960976	0.608978	0.391022	06:44
7	0.863073	1.902601	0.621973	0.378027	06:46
8	0.625353	1.869262	0.621973	0.378027	06:48
9	0.477806	1.833243	0.640284	0.359716	06:49
10	0.361074	1.840771	0.645009	0.354991	06:48
11	0.267574	1.824469	0.650916	0.349084	06:50
12	0.224306	1.835985	0.650325	0.349675	06:56
13	0.178312	1.854152	0.656822	0.343178	06:52

13	0.178312	1.854152	0.656822	0.343178	06:52
14	0.132650	1.832332	0.658004	0.341996	06:59
15	0.110643	1.870082	0.663910	0.336090	06:54
16	0.094084	1.855706	0.664501	0.335499	06:56
17	0.083180	1.837342	0.670998	0.329002	06:49
18	0.068097	1.833766	0.670408	0.329592	06:51
19	0.059121	1.841617	0.671589	0.328411	06:46
20	0.047711	1.812880	0.671589	0.328411	06:54
21	0.041470	1.823810	0.680449	0.319551	06:50
22	0.030779	1.812108	0.685765	0.314235	06:45
23	0.026156	1.822844	0.686356	0.313644	06:47
24	0.023689	1.826892	0.687537	0.312463	06:48
25	0.020368	1.810359	0.685765	0.314235	06:48
26	0.019602	1.803084	0.683993	0.316007	06:48
27	0.015039	1.802364	0.688718	0.311282	06:52
28	0.016286	1.804213	0.689309	0.310691	06:47
29	0.017375	1.803259	0.686946	0.313054	06:54

AdamW (ResNet18, Dataset: g30, 30 epoch)

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	3.323349	2.945581	0.443001	0.556999	07:59
1	2.910522	2.725625	0.473125	0.526875	07:58
2	2.556851	2.516756	0.514471	0.485529	07:58
3	2.169173	2.323439	0.545777	0.454223	07:56
4	1.793148	2.158679	0.570585	0.429415	07:53
5	1.462077	2.014146	0.595393	0.404607	07:52
6	1.158469	1.909761	0.610159	0.389841	07:47
7	0.884748	1.858174	0.621973	0.378027	07:54
8	0.650855	1.826895	0.632605	0.367395	07:51
9	0.472259	1.825389	0.641465	0.358535	07:50
10	0.346873	1.839968	0.644418	0.355582	07:54
11	0.285088	1.865621	0.644418	0.355582	07:55
12	0.212752	1.840978	0.648553	0.351447	07:56

13	0.172265	1.817923	0.661548	0.338452	07:53
14	0.139010	1.834507	0.660366	0.339634	07:52
15	0.121654	1.866984	0.658004	0.341996	07:52
16	0.102217	1.886302	0.657413	0.342587	07:52
17	0.077984	1.878275	0.657413	0.342587	07:54
18	0.069224	1.867965	0.661548	0.338452	07:53
19	0.055382	1.844360	0.668045	0.331955	07:53
20	0.041648	1.829239	0.672770	0.327230	07:50
21	0.036911	1.821233	0.674542	0.325458	07:51
22	0.030286	1.822807	0.675133	0.324867	07:50
23	0.025923	1.825060	0.675133	0.324867	07:46
24	0.019792	1.804646	0.675133	0.324867	07:50
25	0.018370	1.820245	0.672770	0.327230	07:52
26	0.018856	1.803209	0.676314	0.323686	08:39

CONCLUSIONES

Gracias a nuestra investigación pudimos encontrar que el mejor optimizador para el conjunto de datos era AdamW, además de que la arquitectura que mejor capta las diferencias entre hoteles fue Densenet, ya que tuvo una accuracy de .56 en la validación.

REFERENCIAS

He, K., Zhang, X., Ren, S., & Sun, J. (2015, December 10). *Deep residual learning for image recognition*. arXiv.org. Retrieved November 7, 2022, from <https://arxiv.org/abs/1512.03385v1>

Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2016, August 25). *Densely Connected Convolutional Networks*. arXiv.org. Retrieved November 7, 2022, from <https://arxiv.org/abs/1608.06993>

Refugiados, A. C. D. L. N. U. P. L. (s. f.). *Trata de personas*. UNHCR. <https://www.acnur.org/trata-y-trafico-de-personas.html>

Tan, M., & Le, Q. V. (2020, September 11). *EfficientNet: Rethinking model scaling for Convolutional Neural Networks*. arXiv.org. Retrieved November 7, 2022, from <https://arxiv.org/abs/1905.11946v5>

ResNet-50: The Basics and a Quick Tutorial. (2022, October 25). Retrieved November 22, 2022, from Datagen website: <https://datagen.tech/guides/computer-vision/resnet-50/>